1 Лабораторная работа № 1 по курсу дискретного анализа: Строковые алгоритмы

Выполнил студент группы М8О-308б-22 МАИ *Караев Тариел Жоомартбекович.*

1.1 Условие

Необходимо реализовать поиск одного образца в тексте с использованием алгоритма Z-блоков. Алфавит — строчные латинские буквы.

1.2 Метод решения

Сначала строится Z-массив для конкатенации образца и текста, который позволяет эффективно находить совпадения подстроки в тексте. Алгоритм сравнивает символы строки и образца, используя предварительно рассчитанные Z-значения, что обеспечивает линейную сложность поиска.

1.3 Описание программы

В программе реализованы две основные функции:

- calculateZArray — функция, которая вычисляет Z-массив для заданной строки. Z-массив хранит длины наибольших префиксов, совпадающих с подстрокой, начиная с каждой позиции строки. Этот массив используется для быстрого сравнения образца с текстом. Алгоритм работает за линейное время O(n).
- searchPattern — функция, которая выполняет поиск образца в тексте. Она объединяет образец и текст через разделитель $, вычисляет Z-массив для этой конкатенации и находит все вхождения образца в текст. Если длина совпадения в Z-массиве равна длине образца, эта позиция считается вхождением образца в текст.

1.4 Дневник отладки

В основном возникали проблемы с компиляцией в виду неверного выбора комплиятора.

1.5 Тест производительности

Код бенчмарка:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
```

```
#include <chrono>
#include <fstream>
#include "main.cpp"

void benchmarkZFunction(const std::string& txt, const std::string& pat) {
    auto start = std::chrono::high_resolution_clock::now();
    std::vector<int> positions = searchPattern(txt, pat);
    auto end = std::chrono::high_resolution_clock::now();

    std::chrono::duration<double> duration = end - start;
    std::cout << "Z-function search duration: " << duration.count() << " seconds" <<
}

void benchmarkStdSearch(const std::string& txt, const std::string& pat) {
    auto start = std::chrono::high_resolution_clock::now();
    auto it = std::search(txt.begin(), txt.end(), pat.begin(), pat.end());
    auto end = std::chrono::high_resolution_clock::now();

    std::chrono::duration<double> duration = end - start;
    std::cout << "std::search duration: " << duration.count() << " seconds" << std::e

    if (it != txt.end()) {
        std::cout << "Pattern found at position: " << std::distance(txt.begin(), it)
    } else {
        std::cout << "Pattern not found." << std::endl;
    }
}

int main() {
    std::srand(static_cast<unsigned int>(std::time(0)));

    const int TEXT_LENGTH = 100000;
    const int PATTERN_LENGTH = 5;

    std::string txt(TEXT_LENGTH, ' ');
    std::string pat(PATTERN_LENGTH, ' ');

    for (int i = 0; i < TEXT_LENGTH; ++i) {
        txt[i] = 'a' + std::rand() % 26;
    }

    for (int i = 0; i < PATTERN_LENGTH; ++i) {
```

```
        pat[i] = 'a' + std::rand() % 26;
    }

    std::cout << "Generated text and pattern:" << std::endl;
    std::cout << "Text: " << txt.substr(0, 100) << "..." << std::endl;
    std::cout << "Pattern: " << pat << std::endl;

    benchmarkZFunction(txt, pat);
    benchmarkStdSearch(txt, pat);

    return 0;
}
```

Результат работы бенчмарка при длине текста и паттерна, состоящих из случайных символов, 100000 и 5 соответственно:

```
Generated text and pattern:
Text: moezjrwqxysinapjyjgrbkvbefmbumdhajildhdcfvkuxafvlmnoykperegmqlvqxfdaogctdnpapvw
Pattern: ehwat
Z-function search duration: 0.0022268 seconds
std::search duration: 0.0008198 seconds
Pattern not found.
```

Несмотря на теоретическую эффективность алгоритма на основе Z-функции, стандартная функция поиска std::search имеют более оптимизированную реализацию, что позволяет ей работать быстрее.

1.6 Недочёты

В конечном счёте реализация оказалось чуть медленее стандартной std::search.

1.7 Выводы

Реализация поиска образца в тексте с использованием Z-функции продемонстрировала эффективное время выполнения при обработке больших объемов данных, хотя стандартная функция поиска std::search оказалась чуть быстрее. В целом, Z-функция полезна для изучения алгоритмов и понимания работы строковых операций и является довольно эффективным решением задачи поиска подстроки в строке.